In-Place Appends for Real: DBMS Overwrites on Flash without Erase

Sergey Hardock
Databases and Distributed Systems Group
TU-Darmstadt, Germany
hardock@dvs.tu-darmstadt.de

Ilia Petrov
Data Management Lab
Reutlingen University, Germany
ilia.petrov@reutlingen-university.de

Robert Gottstein
Databases and Distributed Systems Group
TU-Darmstadt, Germany
gottstein@dvs.tu-darmstadt.de

Alejandro Buchmann
Databases and Distributed Systems Group
TU-Darmstadt, Germany
buchmann@dvs.tu-darmstadt.de

ABSTRACT

In the present paper we demonstrate a novel approach to handling small updates on Flash called In-Place Appends (IPA). It allows the DBMS to revisit the traditional write behavior on Flash. Instead of writing whole database pages upon an update in an out-of-place manner on Flash, we transform those small updates into update deltas and append them to a reserved area on the very same physical Flash page. In doing so we utilize the commonly ignored fact, that under certain conditions Flash memories can support in-place updates to Flash pages without a preceding erase operation.

The approach was implemented under Shore-MT and evaluated on real hardware. Under standard update-intensive workloads we observed 67% less page invalidations resulting in 80% lower garbage collection overhead, which yields a 45% increase in transactional throughput, while doubling Flash longevity at the same time. The IPA outperforms In-Page Logging (IPL) by more than 50%.

We showcase a Shore-MT based prototype of the above approach, operating on real Flash hardware – the OpenSSD Flash research platform. During the demonstration we allow the users to interact with the system and gain hands-on experience of its performance under different demonstration scenarios. These involve various workloads such as TPC-B, TPC-C or TATP.

1. INTRODUCTION

A well-known property of Flash memory is the erase-before-overwrite principle. In order to update the content of a certain Flash page, the corresponding Flash block must be erased first and all valid pages must be written back. Since this results in huge I/O latencies and rapid wear-out, all modern SSDs utilize some variant of an out-of-place update strategy. The updated Flash pages

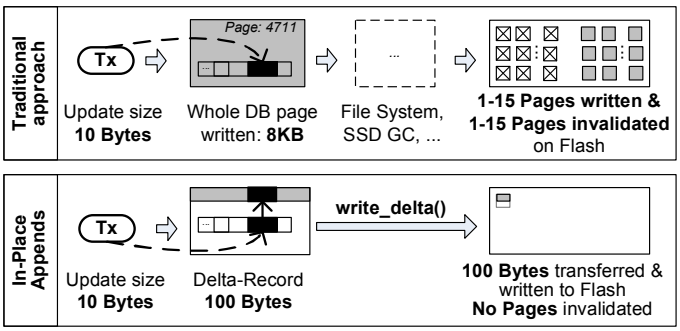

Figure 1: Write-amplification: traditional vs IPA.

are always written to a new physical location, while the old pages are simply invalidated and the occupied space eventually gets reclaimed by the garbage collection (GC). Although, this allows postponing expensive erases and page migrations and executing them in the background, the *on-device write-amplification* produced by the GC is a major performance bottleneck of modern Flash SSDs [4]. Another source of the write-amplification in traditional DBMSs are the *write behavior* and *I/O granularity*. Regardless of the size of the updated information on a database page, the whole page is written out to stable storage (Figure 1). Our analysis of the standard OLTP benchmarks (TPC-B/-C and TATP), as well as social network workload based on LinkBench has shown that in more than 70% of evicted dirty 8KB-pages, less than 100 bytes of net data is modified. Thus, for 100 modified bytes in total the DBMS writes out the whole 8KB database pages. This results in the *DBMS write-amplification* (ratio of written and actually changed bytes) of about 80x. The file system underneath can further increase this value [9].

To handle both kinds of write-amplification on Flash we proposed an approach called *In-Place Appends (IPA)* [5]. Its basic idea is to transform small in-place updates performed by DBMS transactions into delta-records upon page eviction. Furthermore, those delta-records are appended to a reserved area on the *very same physical Flash pages* along with the original content. In doing so we utilize the commonly ignored fact that under certain conditions physical Flash pages can be updated in-place without a

preceding erase operation. By relaxing this *erase-before-overwrite principle* we can significantly reduce the number of page invalidations and out-of-place updates. Furthermore we reduce the GC overhead (page migrations and erase operations) and achieve lower I/O latencies. Additionally, the *DBMS write-amplification* is reduced by a newly defined command *write_delta*, which allows the DBMS to write out only the delta-records instead of whole pages.

The IPA [5] was implemented in Shore-MT. Although the IPA is well applicable to traditional black-box SSD architectures, we have implemented it as an extension of open NoFTL architecture [6], due to the clear performance advantages of the latter. The NSM page layout was accordingly modified to "accommodate" the delta-record area, while buffer and storage management took the responsibility for creating and applying of delta-records for page reconstruction. The use of *NoFTL regions* [7] allows applying IPA selectively, only to certain database objects that are dominated by small-sized updates. The evaluation is performed on the OpenSSD Jasmine hardware: a research SSD platform with programmable controller and MLC Flash modules. Throughout the experiments under standard OLTP workloads (TPC-C, TPC-B and TATP) we observed up to 45% improvement of transactional throughput by performing up to 80% less page migrations and erase operations as compared to the traditional approach. Besides the clear performance advantages, the reduction of GC overhead results in doubling the longevity of Flash SSD.

In-Page Logging [8] is a well-known approach and the closest competitor of IPA. A major difference to IPA is the way the delta-records (or update logs in IPL) are persisted. IPL writes out the update logs either upon the page eviction or fullness of in-memory log buffer. The logs are written to the separate, reserved Flash pages on the same Flash block the original data is. Thus, to reconstruct the up-to-date version of the database page multiple Flash pages must be read (Flash page(s) with the original data and the one or more Flash pages with update logs). Under modern OLTP workloads with 70% to 90% reads, doubling the read load causes significant performance bottlenecks. In contrast, IPA does not produce any additional read overhead, since delta-records are co-located with the original content on the same Flash page. Furthermore, IPA performs 23% to 62% less writes and 29% to 74% less erases as compared to IPL on a range of OLTP workloads.[1]

2. REVISITING ERASE-BEFORE-OVERWRITE PRINCIPLE

The elementary unit of Flash memory is a single Flash cell - a floating gate (or a charge trap in 3D NAND). The cells of each Flash block are connected in the form of a lattice (see Figure 2), where rows are known as wordlines and columns as bitlines. Cells of each wordline build one (SLC) or several (MLC) physical Flash pages. This physical layout of NAND Flash is optimized for the fast access to the whole Flash pages, since writing and reading is done on per wordline basis.

It is worth, however, to look deeper into the write process on the Flash. To program a Flash page, at first, the corresponding wordline (e.g. WL30 on the Figure 2) is selected by applying a high voltage (e.g. 20V) to it. Then, depending on the value of each bit of data being programmed, the voltages on the corresponding bitlines are selected respectively. Thus, for instance, by applying the VCC voltage to a bitline, the corresponding cell on selected wordline is left unprogrammed, i.e. no charge is "inserted" into

[1]The IPL versus IPA comparison was done by using the original IPL simulator and the Flash memory configuration from [8] on traces recorded from running TPC-B/-C and TATP benchmarks.

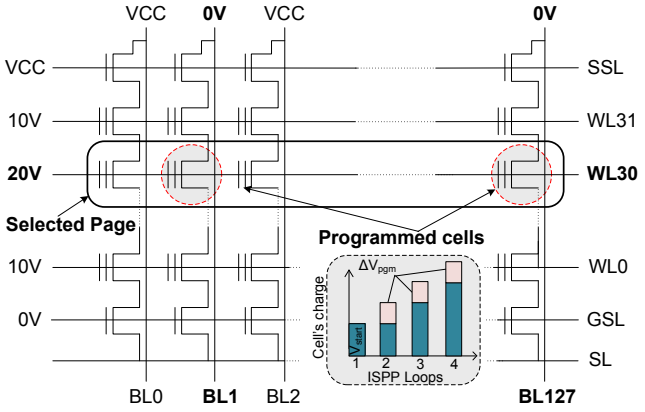

Figure 2: Organization of SLC NAND Flash memory and ISPP.

this cell, while by applying 0V voltage the corresponding cell will be programmed to a certain charge. Further, the programming of each particular cell is done in multiple steps. This technique is applied by all modern Flash SSDs and is known as Incremental Step Pulse Programming (ISPP) [3]. The charge of programmed cells is increased incrementally in small "portions", while after each programming iteration the cell is sensed (read) to check if the desired charge level is achieved. It is important to note, that to increase the charge of any individual cell no foregoing erase operation is required. Only if the charge level needs to be decreased - the whole corresponding Flash block must be erased (i.e. all cells are reseted). The probability that a random update on a Flash page results *only* in increase of the charge levels of corresponding cells[2] is negligibly small. *Therefore, in the common case the updates can not be performed in-place (erase-before-overwrite principle).*

But what if an update on a Flash page is performed in the form of an append? Assume, for instance, the 8KB Flash page is programmed initially with only 6KB data. In this case, the cells that correspond to the remaining 2KB are left unprogrammed. Now, the original 6KB of data are augmented with the 2KB of new data. *This new version of the page (original data & append) can actually be written (programmed) in-place, i.e. by "overwriting" the append area of the original Flash page without foregoing erase operation.* This is possible because all newly programmed cells only increase their charge. The existing charge within the cells storing the original data is left unchanged during the overwrite.

3. BRIEF OVERVIEW OF IN-PLACE APPENDS

The main idea is to transform small-size updates on DB pages into delta-records upon page eviction from the buffer pool. The delta records are then appended to the reserved space on a page, so-called delta-record area, while the original content of the page is left unchanged. By doing so, the database page can be written to the *very same physical Flash address* without page invalidation or foregoing erase operation. The major "points of attack" by the implementation of the approach are: (i) delta-record format, flexible configuration of IPA and database page layout; (ii) DBMS operations - fetching, modification and eviction; (iii) error-correction codes (ECC) on Flash; (iv) program interferences on Flash.

Delta-record, N×M scheme and database page layout. Delta-records store information needed to reconstruct the up-to-date version of the page. Updates are "logged" in byte-granularity,

[2]On SLC Flash this means that all updated bits change from 1 to 0

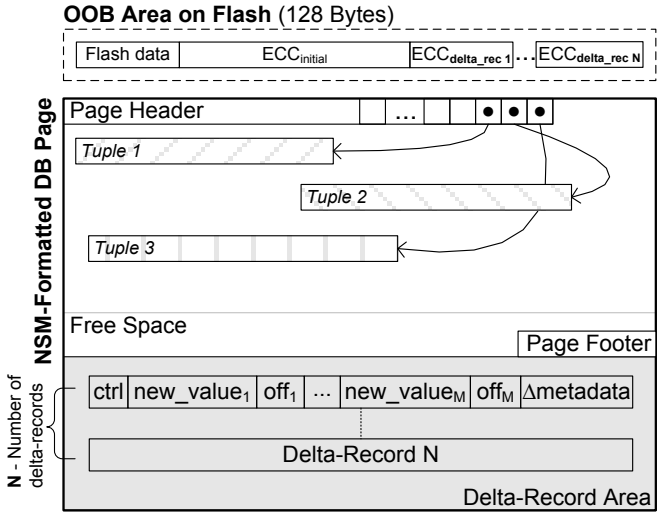

Figure 3: Database page-format, supporting IPA on Flash

i.e. each updated byte is represented in delta-record as a <*new_value, offset*> pair. The configuration parameter M determines the maximum number of such pairs stored in a single delta-record. Furthermore, each delta-record contains: (i) a *control_byte* - a flag representing the presence of the delta-record, and (ii) the modified version of page metadata: header and footer (see Figure 3). To "accommodate" the delta-records on the page we reserve a certain amount of space at the end of the page – the so-called delta-record area. The number of delta-records per page is controlled by the configuration parameter N. Thus, the delta-record area size for a particular *N*×*M* configuration is: $N \times (1 + 3M + \Delta metadata)$.

Page operations. IPA requires certain modifications in the traditional operations on database pages. Before the page is placed into the buffer frame upon being fetched, the storage manager checks if it contains delta-records. If so, those are applied by changing the original bytes at defined offsets to their updated values from the delta-records. Now the page body is in its up-to-date state. Similarly, the page metadata is updated to its actual version from $\Delta metadata$ in the delta-record. Finally, the resulting page is placed into the buffer frame.

When a transaction updates the content of the page, the buffer manager checks if it conforms to the IPA *N*×*M* scheme. Thus, the total number of delta-records (including the existing) cannot exceed N, while the number of changed bytes per delta-record should not exceed M. If those conditions are fulfilled, the update is performed as usual, while the offsets of changed bytes are stored in the delta-record(s). The traditional behavior of the buffer manager is not affected by IPA, since the buffer contains always the up-to-date version of the page, and all updates are done as usually in-place. The violation of one of the above conditions means that upon eviction the page cannot be written out using IPA, and will therefore be written in a traditional out-of-place manner on Flash. In this case, the *out-of-place flag* is set, and further updates are not tracked until eviction.

On page eviction from the buffer pool, the storage manager checks whether the *out-of-place flag* is set. If so, the delta-record area is reset, and the up-to-date version of the page is written out in an out-of-place manner. Otherwise, the page can be overwritten in-place by using in-place appends. In this case, only the delta-record(s) is transmitted to the Flash storage by using the *write_delta()* command.

write_delta(LBA, offset, delta_length, delta_bytes[ ]);

The delta-record(s) will be appended to the very same physical Flash page containing the original database page. This is possible since the original content of the page is left unchanged, while all updates are coalesced in the appended delta-record. *The sole transfer of delta-records (instead of whole pages) significantly reduces the DBMS write-amplification, whereas appending those delta-records to original Flash pages eliminates the need to perform page invalidations and out-of-place writes, which further reduces the GC overhead (on-device write-amplification).* IPA is also applicable to conventional SSDs with block-device interface (see Section 4).

Please note that the regular database functionality (e.g. recovery, locking, etc.) is NOT impacted by the proposed approach. Furthermore, it introduces negligible or no overhead to the DBMS, since (i) it can be selectively applied only to specific database objects using *NoFTL Regions*; (ii) change tracking in the buffer produces min. computational overhead.

Flash types and program interference. In-Place Appends can be applied to all modern types of Flash memory, namely SLC, MLC/eMLC and TLC in 3D NAND. On **SLC** NAND Flash IPA can be applied without specific limitations. The reason is that the difference between different threshold voltages (indicating different logical bit-codes of the Flash cell: 1 and 0) is large enough to compensate small deviations which might appear due to program interference (parasite capacitance-coupling), while (re-)programming the Flash-page (appending the delta-record). The **MLC** Flash is more susceptible to the program interference errors, due to the shorter distances between different voltage thresholds. To safely apply In-place Appends on MLC Flash without increasing program interference we propose two configuration modes. First, the MLC Flash can be used in *pseudo-SLC mode (pSLC)*: the Flash capacity halved as very second page of Flash memory is effectively used (LSB-pages). In this mode the MLC Flash is as tolerant to program interference errors as SLC Flash. Under the second, also called *odd-MLC* mode, the whole MLC Flash capacity is utilized. However, IPA are only applied to LSB pages (odd numbered pages), whereas MSB pages (even numbered pages) still need to be programmed in standard out-of-place manner. **3D NAND** Flash addresses program interference issues by using new manufacturing technologies. According to Samsung their 3D V-NAND chips are: "Bitline Interference Free" and "Wordline Interference Almost Free" [2]. Therefore, IPA is applicable to 3D NAND using the above SLC/pSLC or odd-MLC techniques.

4. DEMONSTRATION

During the demonstration we introduce the audience to basics of the proposed approach and let them evaluate it interactively on real hardware. The demonstration system consists of the Flash storage - the OpenSSD research Flash board[3] connected to a host PC running Shore-MT storage engine (Figure 4). Using an intuitive GUI (Figure 5) the audience can configure a sequence of tests and experience live the performance advantages of the IPA. The proposed demonstration scenarios are as follows.

Demo-Scenario 1 – Baseline.

The audience picks one of the three available OLTP benchmarks

[3]Four dual-die Samsung K9LCG08U1M 8GB packages per module. Each package consists of 4096 erase units each holding 128 16KB Flash pages [1].

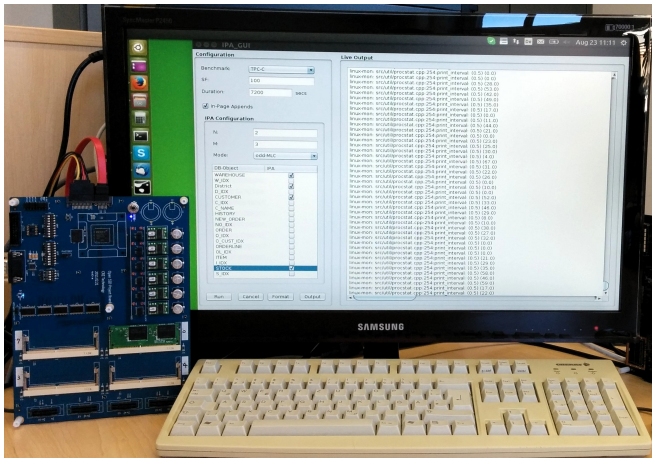

Figure 4: Demonstration system with the OpenSSD board.

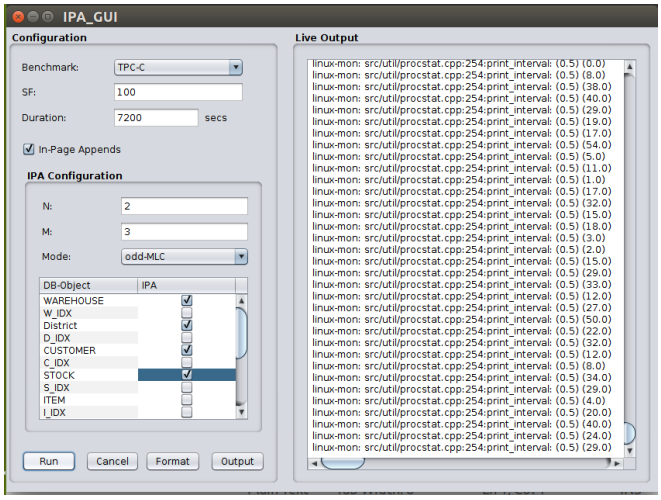

Figure 5: GUI for the evaluation of IPA.

(TPC-B, TPC-C or TATP), selects the desired scaling factor (limited by 64GB of Flash storage) and the duration of the test. The DBMS executes the benchmark using the traditional approach as a baseline, i.e. every updated DB-page results in one or more out-of-place writes on Flash. During the benchmark run the audience can observe the current transactional throughput. At the end detailed statics of performed I/Os are visualized.

Demo-Scenario 2 – IPA for conventional SSD.

In this scenario the audience examines IPA designed for conventional SSDs. Its implementation assumes the use of traditional block-device interface. The DBMS writes out whole pages in the format: page body + delta-record area. After the main parameters of IPA have been selected ($N \times M$ scheme and the mode of IPA on MLC Flash: pSLC or odd-MLC), and the Flash SSD is completely formatted (low-level formatting) the benchmark is run with the same scaling factor and for the same duration as in the baseline test. The audience can compare the output results of both approaches (throughput, I/O statistics).

Demo-Scenario 3 – IPA for native Flash.

This scenario is similar to the previous one, however, the DBMS utilizes IPA designed for native Flash (e.g. NoFTL architecture). In this case only the delta-records are transferred to the Flash storage. Both IPA scenarios #2 and #3 result in the same reduction of GC overhead, since in both cases updates are performed as in-place appends reducing the number of page invalidations. However, here IPA uses *write_delta* command, which significantly reduces the DBMS write-amplification and the amount of transferred data.

Table 1: TPC-B: traditional approach (no In-Place Appends [0×0]) vs. [2×4] scheme in modes pSLC and odd-MLC.

	0x0 Absolute	2x4 Absolute *pSLC*	2x4 Relative *pSLC* [%]	2x4 Absolute *odd-MLC*	2x4 Relative *odd-MLC* [%]
Out-of-Place Writes vs. In-Place Appends			33/67		51/49
Host Reads (16KB)	3 779 926	5 540 034	+47	4 875 961	+29
Host Writes (16KB)	2 028 626	3 047 538	+50	2 372 017	+17
GC Page Migrations	605 047	153 201	-75	315 228	-48
GC Erases	15 839	7 401	-53	7 625	-52
Page Migrations per Host Write	0.2983	0.0503	-83	0.1329	-55
GC Erases per Host Write	0.0078	0.0024	-69	0.0032	-59
Transactional Throughput	260	380	+46	313	+20

Table 1 shows the comparison results of TPC-B benchmark running for two hours on OpenSSD board (during the demonstration the durations of 5 or 10 minutes are sufficient for a comparison). The experiments were performed (i) without IPA ([0×0] column), and with IPA using (ii) pSLC and (iii) odd-MLC modes with [2×4] configuration scheme. Under TPC-B, IPA outperforms the traditional approach by executing up to 70% less erases and up to 85% less page migrations. This reduction of GC overhead has two major advantages: (i) the increase of the transactional throughput of up to 45%, and (ii) doubling the Flash SSD lifetime.

Acknowledgments

This paper was supported by the German BMBF "Software Campus" (01IS12054) and the German Research Foundation (DFG) project "Flashy-DB".

5. REFERENCES

[1] The openssd project. http://www.openssd-project.org, 2014.
[2] Samsung v-nand. http://www.samsung.com/us/business/oem-solutions/pdfs/V-NAND_technology_WP.pdf, 2014.
[3] S. Aritome. *NAND flash memory technologies*. IEEE Press series on microelectronic systems. Wiley-IEEE Press, 2016.
[4] F. Chen, D. A. Koufaty, and X. Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *Proc. SIGMETRICS'09*.
[5] S. Hardock, I. Petrov, R. Gottstein, and A. Buchmann. From in-place updates to in-place appends: Revisiting out-of-place updates on flash. In *Proc. SIGMOD'17*.
[6] S. Hardock, I. Petrov, R. Gottstein, and A. Buchmann. Noftl: Database systems on ftl-less flash storage. In *Proc. VLDB'13*.
[7] S. Hardock, I. Petrov, R. Gottstein, and A. Buchmann. Revisiting dbms space management for native flash. In *Proc. EDBT*, 2016.
[8] S.-W. Lee and B. Moon. Design of flash-based dbms: An in-page logging approach. In *Proc. SIGMOD'07*.
[9] Y. Lu, J. Shu, and W. Zheng. Extending the lifetime of flash-based storage through reducing write amplification from file systems. In *Proc. FAST'13*.